# Лабораторная работа №3
# Решение задач на собственные значения

Козлова Анастасия Олеговна
Группа: 10
Вариант: 6

## 1 Постановка задачи

### 1.1 Общая постановка

1. Сгенерировать симметричную матрицу $A$ размерности $n$ ($n$ от 10 до 15)
2. Матрица не должна быть диагональной или трехдиагональной
3. Сохранить сгенерированную матрицу
4. Реализовать для матрицы указанные в варианте методы
5. Точность вычислений: $\epsilon = 10^{-4}$

### 1.2 Вариант 6

- **Задача 1:** С помощью QR-алгоритма (без сдвига) найти все собственные значения матрицы $A$, предварительно приведя ее к верхней форме Хессенберга
- **Задача 2:** С помощью степенного метода найти наибольшее по модулю собственное значение и соответствующий ему собственный вектор
- **Выходные данные:**
  1. Количество выполненных итераций
  2. Полученные собственное значение и собственный вектор $(\lambda_1, x_1)$
  3. Значение $\|Ax_1 - \lambda_1 x_1\|$
- **Критерий остановки для степенного метода:**

$$\|Au^{(k)} - \lambda^{(k)} u^{(k)}\| < \epsilon$$

где $u^{(k)}$, $\lambda^{(k)}$ – собственный вектор и собственное значение, полученные на $k$-й итерации

# 2 Теоретическая часть

## 2.1 QR-алгоритм с приведением к форме Хессенберга

**Этапы алгоритма:**

1. **Приведение к форме Хессенберга** методом Хаусхолдера:

$$H = Q^T A Q$$

где $H$ – матрица в форме Хессенберга (нули ниже первой поддиагонали), $Q$ – ортогональная матрица.

2. **QR-разложение матрицы Хессенберга** методом вращений Гивенса:

$$H = QR$$

3. **Итерационный процесс QR-алгоритма**:

$$\begin{aligned} H_1 &= H \\ H_k &= Q_k R_k \\ H_{k+1} &= R_k Q_k \end{aligned}$$

4. **Собственные значения** извлекаются из диагонали матрицы $H_k$ при $k \to \infty$.

## 2.2 Степенной метод

Итерационный процесс для нахождения наибольшего по модулю собственного значения:

1. **Инициализация:** $u^{(0)} = (1, 1, \ldots, 1)^T$, нормировка: $u^{(0)} = \frac{u^{(0)}}{\|u^{(0)}\|}$
2. **Итерационный шаг:**

$$\begin{aligned} v^{(k)} &= A u^{(k-1)} \\ \lambda^{(k)} &= \frac{(u^{(k-1)})^T v^{(k)}}{(u^{(k-1)})^T u^{(k-1)}} \quad \text{(отношение Рэлея)} \\ u^{(k)} &= \frac{v^{(k)}}{\|v^{(k)}\|} \end{aligned}$$

3. **Критерий остановки:**

$$\|A u^{(k)} - \lambda^{(k)} u^{(k)}\| < \epsilon$$

# 3 Реализация методов

## 3.1 Структура программы

Программа разделена на следующие файлы:

- `matrix.h, matrix.cpp` – класс для работы с матрицами
- `eigen_solver.h, eigen_solver.cpp` – функции для вычисления собственных значений
- `main.cpp` – основная программа с тестированием

## 3.2 Ключевые методы

```
Matrix hessenbergForm(const Matrix& A) {
    int n = A.size();
    Matrix H = A;

    for (int k = 0; k < n - 2; k++) {
        double sigma = 0.0;
        for (int i = k + 1; i < n; i++) {
            sigma += H(i, k) * H(i, k);
        }

        if (sigma < 1e-20) continue;

        double alpha = sqrt(sigma);
        if (H(k + 1, k) < 0) alpha = -alpha;

        std::vector<double> v(n - k - 1, 0.0);
        v[0] = H(k + 1, k) + alpha;

        for (int i = k + 2; i < n; i++) {
            v[i - k - 1] = H(i, k);
        }

        double vtv = 0.0;
        for (double vi : v) vtv += vi * vi;

        if (vtv < 1e-20) continue;
        double beta = 2.0 / vtv;

        for (int j = k; j < n; j++) {
            double dot = 0.0;
            for (int i = 0; i < n - k - 1; i++) {
                dot += v[i] * H(k + 1 + i, j);
            }
            for (int i = 0; i < n - k - 1; i++) {
                H(k + 1 + i, j) -= beta * v[i] * dot;
            }
        }

        for (int i = 0; i < n; i++) {
            double dot = 0.0;
            for (int j = 0; j < n - k - 1; j++) {
                dot += H(i, k + 1 + j) * v[j];
            }
            for (int j = 0; j < n - k - 1; j++) {
                H(i, k + 1 + j) -= beta * dot * v[j];
            }
        }
    }

    for (int i = 2; i < n; i++) {
        for (int j = 0; j < i - 1; j++) {
```

```
            H(i, j) = 0.0;
        }
    }

    return H;
}
```

Листинг 1: Приведение к форме Хессенберга (метод Хаусхолдера)

```
std::pair<Matrix, Matrix> qrDecompositionHessenberg(const Matrix& H) {
    int n = H.size();
    Matrix Q(n), R = H;

    for (int i = 0; i < n; i++) Q(i, i) = 1.0;

    for (int i = 0; i < n - 1; i++) {
        double a = R(i, i), b = R(i + 1, i);
        double r = sqrt(a * a + b * b);

        if (fabs(r) < EPS) continue;

        double c = a / r, s = -b / r;

        for (int j = i; j < n; j++) {
            double temp1 = R(i, j), temp2 = R(i + 1, j);
            R(i, j) = c * temp1 - s * temp2;
            R(i + 1, j) = s * temp1 + c * temp2;
        }
        R(i + 1, i) = 0.0;

        for (int j = 0; j < n; j++) {
            double temp1 = Q(j, i), temp2 = Q(j, i + 1);
            Q(j, i) = c * temp1 - s * temp2;
            Q(j, i + 1) = s * temp1 + c * temp2;
        }
    }

    return std::make_pair(Q, R);
}
```

Листинг 2: QR-разложение матрицы Хессенберга (метод Гивенса)

```
std::pair<std::vector<double>, int> qrAlgorithm(const Matrix& A) {
    int n = A.size();
    Matrix H = hessenbergForm(A);
    Matrix Ak = H;
    int iterations = 0;
    const int maxIterations = 1000;

    std::vector<double> prevDiagonal(n, 0.0);
    for (int i = 0; i < n; i++) prevDiagonal[i] = Ak(i, i);

    while (iterations < maxIterations) {
        auto [Q, R] = qrDecompositionHessenberg(Ak);
```

```
13
14          Matrix nextAk(n);
15          for (int i = 0; i < n; i++) {
16              for (int j = 0; j < n; j++) {
17                  double sum = 0.0;
18                  for (int k = 0; k < n; k++) {
19                      sum += R(i, k) * Q(k, j);
20                  }
21                  nextAk(i, j) = sum;
22              }
23          }
24
25          Ak = nextAk;
26          iterations++;
27
28          double maxChange = 0.0;
29          for (int i = 0; i < n; i++) {
30              double change = fabs(Ak(i, i) - prevDiagonal[i]);
31              maxChange = std::max(maxChange, change);
32              prevDiagonal[i] = Ak(i, i);
33          }
34
35          double maxSubDiag = 0.0;
36          for (int i = 1; i < n; i++) {
37              maxSubDiag = std::max(maxSubDiag, fabs(Ak(i, i-1)));
38          }
39
40          if ((maxSubDiag < EPS * 10) ||
41              (maxChange < EPS * 10 && iterations > 10)) {
42              break;
43          }
44      }
45
46      std::vector<double> eigenvalues(n);
47      for (int i = 0; i < n; i++) {
48          eigenvalues[i] = Ak(i, i);
49      }
50
51      std::sort(eigenvalues.begin(), eigenvalues.end(),
52          [](double a, double b) { return fabs(a) > fabs(b); });
53
54      return std::make_pair(eigenvalues, iterations);
55  }
```

Листинг 3: QR-алгоритм без сдвига

```
1   std::pair<std::pair<double, std::vector<double>>, int>
2   powerMethod(const Matrix& A) {
3       int n = A.size();
4       int iterations = 0;
5       const int maxIterations = 10000;
6
7       std::vector<double> x(n, 1.0);
8       x = normalize(x);
```

```
9
10      double lambda = 0.0;
11      double lambdaPrev = 0.0;
12      double error = 1.0;
13
14      while (iterations < maxIterations && error >= EPS) {
15          lambdaPrev = lambda;
16
17          std::vector<double> Ax = A.multiply(x);
18
19          double numerator = 0.0, denominator = 0.0;
20          for (int i = 0; i < n; i++) {
21              numerator += x[i] * Ax[i];
22              denominator += x[i] * x[i];
23          }
24          lambda = numerator / denominator;
25
26          double norm = vectorNorm(Ax);
27          for (int i = 0; i < n; i++) {
28              x[i] = Ax[i] / norm;
29          }
30
31          iterations++;
32
33          std::vector<double> AxCurrent = A.multiply(x);
34          error = 0.0;
35          for (int i = 0; i < n; i++) {
36              double diff = AxCurrent[i] - lambda * x[i];
37              error += diff * diff;
38          }
39          error = sqrt(error);
40      }
41
42      std::vector<double> AxFinal = A.multiply(x);
43      double lambdaFinal = 0.0;
44      double normXFinal = 0.0;
45      for (int i = 0; i < n; i++) {
46          lambdaFinal += x[i] * AxFinal[i];
47          normXFinal += x[i] * x[i];
48      }
49      lambdaFinal /= normXFinal;
50
51      return std::make_pair(std::make_pair(lambdaFinal, x), iterations);
52  }
```

Листинг 4: Степенной метод

# 4 Результаты вычислений

## 4.1 Выходные данные программы

```
================================================
Lab 3: Eigenvalue Problems Solution
```

```
================================================

1. Generating symmetric matrix A of size 12
Matrix A saved to file matrix_A.txt

First 5 rows and columns of matrix A:
       4.1        3.8        5.3       0.12       0.11
       3.8        0.7        8.8          6       0.18
       5.3        8.8        6.7        7.2        3.8
      0.12          6        7.2        4.6          1
      0.11       0.18        3.8          1        8.3

2. QR algorithm (without shift) with Hessenberg form reduction
------------------------------------------------
Number of iterations: 196
Found eigenvalues:
1 = 45.208060
2 = 20.003673
3 = -16.678857
4 = 9.951574
5 = 7.853218
6 = -7.684354
7 = 6.435776
8 = -5.534195
9 = -3.139364
10 = 2.702626
11 = 0.509429
12 = -0.327585

3. Power method
------------------------------------------------
Number of iterations: 9
Largest eigenvalue by absolute value:  = 45.208059
Corresponding eigenvector (first 5 components):
x[0] = 0.042333
x[1] = 0.076539
x[2] = 0.133300
x[3] = 0.286438
x[4] = 0.407366

4. Results comparison
------------------------------------------------
Largest eigenvalue by QR algorithm: 45.208060
Largest eigenvalue by power method: 45.208059
Difference: 0.000001

5. Accuracy check for power method
------------------------------------------------
||A*x - *x|| = 0.000046
Required precision  = 0.000100
```

```
Precision achieved!
```

# 5 Анализ результатов

## 5.1 Генерация матрицы

- Сгенерирована симметричная матрица размерности $n = 12$
- Матрица не является диагональной или трехдиагональной
- Матрица сохранена в файл `matrix_A.txt`

## 5.2 QR-алгоритм

- Количество итераций: 196
- Найдены все 12 собственных значений
- Наибольшее собственное значение по модулю: $\lambda_1 = 45.208060$
- Алгоритм корректно сошелся к квазитреугольной форме

## 5.3 Степенной метод

- Количество итераций: 9
- Найдено наибольшее собственное значение: $\lambda = 45.208059$
- Получен соответствующий собственный вектор
- Точность достигнута: $\|Ax - \lambda x\| = 0.000046 < \epsilon = 0.0001$
- Метод показал высокую скорость сходимости

## 5.4 Сравнение методов

- **Совпадение результатов:**

$$\begin{aligned} \text{QR: } \lambda_1 &= 45.208060 \\ \text{Power: } \lambda &= 45.208059 \\ \text{Разница: } \Delta &= 0.000001 \end{aligned}$$

- **Скорость сходимости:**
  - Степенной метод: 9 итераций
  - QR-алгоритм: 196 итераций
- **Точность:**
  - Степенной метод достиг заданной точности $\epsilon = 10^{-4}$
  - Проверка точности: $\|Ax - \lambda x\| = 4.6 \times 10^{-5}$

### 5.5 Проверка точности

- Вычисленная величина $\|Ax_1 - \lambda_1 x_1\| = 0.000046$
- Заданная точность $\epsilon = 0.0001$
- Условие точности выполнено: $0.000046 < 0.0001$
- Собственный вектор корректно соответствует найденному собственному значению

## 6 Выводы

1. **Корректность реализации:** Оба алгоритма (QR и степенной метод) корректно реализованы и дают согласованные результаты
2. **Точность вычислений:** Степенной метод достиг требуемой точности $\epsilon = 10^{-4}$ за 9 итераций
3. **Сравнение эффективности:**
   - **Степенной метод** более эффективен для нахождения **наибольшего** собственного значения (9 итераций)
   - **QR-алгоритм** позволяет найти **все** собственные значения, но требует больше итераций (196)
4. **Сходимость алгоритмов:**
   - QR-алгоритм демонстрирует линейную сходимость
   - Степенной метод показывает быструю сходимость благодаря отношению Рэлея
5. **Верификация результатов:** Совпадение наибольших собственных значений, полученных разными методами ($\Delta = 10^{-6}$), подтверждает корректность вычислений
6. **Качество собственного вектора:** Малая величина $\|Ax - \lambda x\|$ ($4.6 \times 10^{-5}$) свидетельствует о хорошей точности найденного собственного вектора

**Рекомендации по использованию методов:**

- Для нахождения **наибольшего** собственного значения рекомендуется использовать **степенной метод** как более быстрый
- Для нахождения **всех** собственных значений необходимо использовать **QR-алгоритм**
- Приведение к форме Хессенберга существенно ускоряет QR-алгоритм для симметричных матриц
- Использование отношения Рэлея в степенном методе повышает точность вычисления собственного значения

Реализованные алгоритмы успешно решают поставленные задачи нахождения собственных значений. Корректность реализации подтверждается согласованностью результатов, достижением заданной точности и выполнением проверочных соотношений.